Propagating Ideal Kernel Eigen-function for Semi-supervised Learning

Abstract

Semi-supervised kernel is a useful tool towards lending the abundant theories and algorithms in kernel methods to semi-supervised learning. Currently, a large family of algorithms relies on spectral transformations, i.e., the new kernel is built on the eigenvectors of the empirical kernel matrix (such as graph Lapacian) with rectified eigenvalues. Though demonstrating lot of success, these algorithms are computationally demanding, and the empirical kernel eigenvectors can be inaccurate due to various practical factors. This paper pursues a new direction in which the desired eigenvectors are computed via propagating the eigenfunction of ideal kernel. The extrapolation builds upon important graph structures encoded in the data, and easily takes into account the resultant matching with the ideal kernel. The inclusion of supervised information in computing the eigenvectors, on the other hand, make them more reliable and less dependent on the choice of empirical kernels. Our algorithm demonstrates improved target alignment and generalization performance. It requires only linear time and space consumptions, and is empirically orders of magnitudes times faster than the counterpart algorithms.

1 Introduction

Semi-supervised learning (SSL) is a useful learning scenario where availability of large volumes of unlabeled samples is shown to boost the learning performance with only limited supervision. Among the various directions that have been pursued on this topic, semi-supervised kernel design turns to a promising one because it allows the abundant theories and algorithms in kernel methods to be lent directly in solving SSL problem. A large family of algorithms for semi-supervised kernel relies on spectral transformation, where the eigenvectors of the empirical kernel matrix (mostly notably the graph Lapacian) are used together with the rectified eigenvalues to build the new kernel.

Given an $n \times n$ similarity matrix K (or kernel matrix), the graph Lapacian is computed as $\mathcal{L} = D - W$, where $D \in \mathbb{R}^{n \times n}$ is a (diagonal) degree matrix such that $D_{ii} = \sum_{j=1}^{n} K_{ij}$. The normalized graph Lapacian is defined as $\tilde{\mathcal{L}} = \mathbf{I} - D^{-1/2} K D^{-1/2}$, where $\mathbf{I}$ is identity matrix. The (normalized) graph Lapacian matrix imposes important smoothness constraints over the graph. In particular, a smaller eigenvalue of it corresponds to a smoother eigenvector over the graph, i.e., the entries of the eigenvector corresponding to neighboring sample points are close to each other. Such smoothness is very useful for predicting the actual class labels. Based on this property, a general principle is applied in spectral transformation to build semi-supervised kernel [7],

$$\tilde{K} = \sum_{i=1}^{n} r(\lambda_i) \phi_i \phi_i^{\top}. \tag{1}$$

Here λ_i's ($i = 1, 2, ..., n$) are eigenvalues of the graph Lapacian $\mathcal{L} \in \mathbb{R}^{n \times n}$ sorted in an ascending order, ϕ_i's are the corresponding eigenvectors, and $r(\cdot)$ is a non-increasing function which enforces larger penalty for less smooth eigenvectors. Various choice of the transform r has been proposed in the literatures. For example, the diffusion kernel [4] corresponds to $r(\lambda) = \exp(-\frac{\sigma^2}{2}\lambda)$; the Gaussian filed kernel [8] uses $r(\lambda) = \frac{1}{\lambda + \epsilon}$. For cluster kernel [1], the eigenvectors ϕ_i's are based on the degree-normalized kernel matrix $\mathcal{S} = D^{-1/2} K D^{-1/2}$. Since $\mathcal{S}$ and $\tilde{\mathcal{L}}$ have the same set of eigenvectors, with the corresponding eigenvalues (pair) adding up to 1, $r(\cdot)$ is chosen as a non-decreasing function in the cluster kernel, such as the linear $r(\lambda) = \lambda$, the step $r(\lambda) = \lambda_i$ if $i \leq k$, and $r(\lambda_i) = 0$ if $i > 0$; or the polynomial $r(\lambda) = \lambda^d$.

Recently, the empirical kernel alignment [2] is proposed to evaluate the fitness of a kernel to the class labels via the use of "ideal kernel" $K(\mathbf{x}, \mathbf{z}) = y(\mathbf{x})y(\mathbf{z})$, where $y(\mathbf{x})$ is the target concept (the class label). It has been shown that the alignment score is sharply concentrated and has a close connection with the generalization performance of a classifier [2]. Therefore maximizing the alignment with the ideal kernel proves a general and effective way for semi-supervised kernel design. For example, a semi-definite programming formulation has been proposed in [6] to learn a kernel matrix $\tilde{K}$ that is maximally aligned with the ideal kernel

$$\max_{\tilde{K}} \langle \tilde{K}_{tr}, \mathbf{y}\mathbf{y}^\top \rangle, \quad \text{subject to} \|\tilde{K}\|_F = 1, \tilde{K} \succeq 0, \ \text{trace}(\tilde{K}) = 1. \tag{2}$$

Here $\tilde{K}_{tr}$ is the block of $\tilde{K}$ corresponding to the training samples, $\mathbf{y}$ is the vector of training labels. In particular, if the kernel matrix $\tilde{K}$ is restricted to be spanned by the eigenvectors of the Laplacian, i.e., $\tilde{K} = \sum_{i=1}^{n} \mu_i \phi_i \phi_i^\top$, then the formulation (2) will be reduced to a quadratically constrained quadratic programming (QCQP) [6, 5]. This avoids the need to choose parameters in $r(\cdot)$, and the nonparametric transform will lead to more flexible kernels. In [5], an order constraint on the transformed eigenvalue is further considered, which leads to the following optimization problem:

$$\begin{aligned} \max_{\mu} \quad & \text{vec}(\mathbf{y}\mathbf{y}^\top)^\top M\mu \\ \text{subject to} \quad & \|M\mu\| \leq 1, \ \mu_i \geq 0, \\ & \mu_i \geq \mu_{i+1}, i = 1, 2, ..., n-1. \end{aligned}$$

Here μ is the vector of transformed eigenvalues μ_i's, $\text{vec}(\cdot)$ is the column vectorization operator of a matrix, $M = [\text{vec}(K_{1,tr}), ..., \text{vec}(K_{m,tr})]$ where $K_{i,tr}$ is the sub-block of $K_i = \phi_i \phi_i^\top$ corresponding to the labeled samples, and ϕ_i's are the eigenvectors of the graph Lapacian sorted in ascending order based on their corresponding eigenvalues. The order constraint reflects important prior belief that smoother eigenvectors should be given higher priority in building the kernel, and empirically it has shown improved behavior over parametric and purely nonparametric spectral transforms [5].

Lots of empirical successes have been observed with the family of semi-supervised kernels based on spectral transform. However, the optimization procedure involved can be quite expensive. For example, computing the eigenvalue decomposition of the Lapacian already takes $O(n^3)$ time and $O(n^2)$ memory, and the complexity of QCQP and SDP is even more demanding. Another concern is that building a kernel solely based on transform of the spectrum may be restrictive in terms of acquiring the desired kernel. Note that, eigenvectors have $n(n-1)$ degree of freedom while eigenvalues only have n, therefore the eigenvectors play a larger role in "shaping" the kernel matrix. However, eigenvectors of empirical kernel matrix (such as the graph Laplacian) can be inaccurate due to various practical factors such as noise, kernel parameters, or class separability. If these eigenvectors are simply left intact, their intrinsic flaws would inevitably inherit in building the new kernel, leading to hampered alignment with the target concept and henceforth degenerated performance.

To solve these problems, in this paper we proposed a new way for designing semi-supervised kernels. We do not grab the eigenvectors directly from the graph Lapacian; actually we do not even attempt to learn a transform on them (for target alignment) because intrinsic flaws would still persist. Instead, a set of desired eigenvectors are newly computed by propagating ideal kernel eigenfunctions to the overall data. The extrapolation builds upon important graph structures encoded in the input patterns, and at the same time takes into account the resultant matching with the ideal kernel. Due to the incorporation of the supervised information in computing the eigenvectors, we observe improved target alignment and generalization performance. The proposed method is very efficient, where the eigenvectors needed only depends on the number of classes. It scales linearly with the sample size and dimension, and runs orders of magnitudes times faster than existing approaches.

The rest of the paper is organized as follows. In section 2, we analyze properties of the ideal kernel matrix and introduce the concept of kernel eigenfunction extrapolation. Based on these background, in section 3 we proposed a new algorithm for semi-supervised kernel design by propagating the ideal kernel eigenfunctions. In section 4, we compare the performance of our approach with a number of algorithms based on spectral transformations. The last section concludes the paper.

2 Ideal Kernel and Eigenfunction Extrapolation

In this section we give a detailed analysis on the eigen-system of the kernel matrix. First, we show that the eigenvectors of the ideal kernel provides a piecewise constant embedding which perfectly separates different classes. Although this is for the ideal situation, it provides a generally useful insight on the "discriminating roles" of the kernel matrix. Then we introduce some background on the kernel eigenfunction and its extrapolation property, which will be used in our algorithm.

2.1 Ideal Kernel

Kernel matrix is the basic building block of kernel machines, which describes the pairwise sample similarities. Here we examine how the kernel matrix provides the discrimination crucial for learning. For simplicity, we consider the ideal kernel. But the intuition applies to general kernel matrix as well. In a multi-class problem, the ideal kernel matrix is block-diagonal as follows

$$K^* = \begin{bmatrix} 11'_{n_1} & 0 & \dots & 0 \\ 0 & 11'_{n_2} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & 11'_{n_k} \end{bmatrix}. \tag{3}$$

Here n_k is the number of samples in the kth class, and we have assumed that samples are ordered such that the first n_1 samples belong to the first class, the following n_2 samples belong to the second class, and so on. The eigen-system of the ideal kernel K^* has the following properties.

Proposition 1. *Given the ideal kernel K^* (3) with C clusters and corresponding cluster size n_k's;*

- *the dominant C eigenvalues are non-zero, each corresponding to the size of one class; the rest $n - C$ eigenvalues are zeros.*
- *the kth eigenvector $\mathbf{v}_k^*$ ($1 \le k \le C$) is* $\mathbf{v}_k^*(i) = \begin{cases} \frac{1}{\sqrt{n_k}} & \sum_{k=1}^{i-1} n_k + 1 \le i \le \sum_{k=1}^{i} n_k \\ 0 & otherwise \end{cases}$.

Proof. Let the eigenvalue decomposition of the ideal kernel matrix be $K^*\mathbf{v}^* = \lambda^*\mathbf{v}^*$. Note that K^* only has C different rows (orthogonal to each other). Therefore it has rank C with $n - C$ zero eigenvalues. On the other hand, note that the ith entry of v equals $\frac{1}{\lambda^*}K^*(i,:)\mathbf{v}^*$, and K^* has a block-wise constant structure. Therefore $\mathbf{v}^*$ is piecewise constant. Let $\mathbf{v}^*$ be written as $\mathbf{v}^* = [\underbrace{v_1, ..., v_1}_{n_1}\underbrace{v_2, ..., v_2}_{n_2}, ..., \underbrace{v_k, ..., v_k}_{n_k}]'$. Then the eigensystem can be written as a equation group

$$\begin{cases} n_1 v_1 &= \lambda^* v_1 \\ n_2 v_2 &= \lambda^* v_2 \\ \dots & \dots \quad \dots \\ n_k v_k &= \lambda^* v_k \end{cases} \tag{4}$$

Each equation in (4) gives rise to two conditions: $\lambda = n_k$, or $v_k = 0$. For the whole equation groupto hold true, however, it's impossible to set $\lambda = n_k$ for $k = 1, 2, ..., C$, since the size of different classes can be different. The only feasible way is to set λ equal to one of the n_k's, $\lambda = n_{k_0}$, and at the same time set $v_k = 0$ for all the $k \neq k_0$. There are C different ways to choose k_0, i.e., for each $k_0 = 1, 2, ..., C$, the eigenvalue $\lambda = n_{k_0}$; as to the eigenvector, all the entries corresponding to class k ($k \neq k_0$) will be zero, and entries corresponding to class k_0 will be $\frac{1}{\sqrt{n_{k_0}}}$ (since they are equal and should normalize to 1). This completes the proof. □

Proposition 1 shows that the eigenvectors of the ideal kernel corresponding to the dominant C eigenvalues will map different classes onto C points lying exactly on the C orthogonal coordinate axes, which provides the strongest discriminating information. On the other hand, the eigenvalues only signify the "strength" of the Cth cluster. Actually, even the eigenvalues were changed arbitrarily (under positivity constraint), the resultant (ideal) kernel matrix is still block diagonal, and can be used to produce exactly the same classification result when fed to in a kernel machine. In other words, the discrimination powers of the kernel matrix is largely manifested by its eigenvectors.

Although the eigenvectors of the ideal kernel provide very useful discriminating information, it is only available on the training examples. Therefore a natural path is to "expand" this ideal information to the whole data set. As will be discussed in the next part, this can be achieved based on the eigenfunction extrapolation. Note that based on the ideal kernel, we can also compute the ideal graph Lapacian $\mathcal{L}^* = D^* - K^*$, where $D^* \in \mathbb{R}^{m\times m}$ is the degree matrix computed based on K^*, or the normalized one $\tilde{\mathcal{L}}^* = \mathbf{I} - (D^*)^{-1/2}K^*(D^*)^{-1/2}$. Due to the block-wise constant nature of K^*, the eigenvectors of $\mathcal{L}^*$ and $\tilde{\mathcal{L}}^*$ are also piecewise constant and provide perfect embedding for class discrimination (detailed derivations are similar to those for the ideal kernel and skipped here). In other words, they have the same discriminating power as the ideal kernel matrix. Therefore in this paper we simply focus on the ideal kernel matrix.

2.2 Eigenfunction Expansion

Let $\mathcal{A}$ be a linear operator on a function space. The eigenfunction f of a linear operator is any non-zero function in that space that returns itself from the operator except for a multiplicative scaling factor, i.e., $\mathcal{A}f = \lambda f$, where λ is called the eigenvalue. In this paper, we are interested in the case where $\mathcal{A}$ is a symmetric, positive semi-definite kernel $K(\mathbf{x}, \mathbf{z})$. The corresponding eigenfunction $\phi(\cdot)$, given the underlying sample distribution $p(\mathbf{x})$, is defined as

$$\int K(\mathbf{x}, \mathbf{z})\phi(\mathbf{x})p(\mathbf{x})d\mathbf{x} = \lambda\phi(\mathbf{z}). \tag{5}$$

The standard numerical method to approximate the eigenfunctions and eigenvalues in (5) is to replace the integral with the empirical average

$$\int K(\mathbf{x}, \mathbf{z})p(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} \approx \frac{1}{n}\sum_{i=1}^{n} K(\mathbf{x}_i, \mathbf{z})\phi(\mathbf{x}_i). \tag{6}$$

By choosing $\mathbf{z} = \mathbf{x}_i$ for $i = 1, 2, ..., n$, equation (6) extends to a matrix eigenvalue decomposition $K\mathbf{v} = \lambda\mathbf{v}$, where K is the kernel matrix defined as $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ for $1 \le i, j \le n$, and $\mathbf{v}$ is the discrete counterpart of ϕ in that $\phi(\mathbf{x}_i) \approx \mathbf{v}(i)$. Then the eigenfunction can be extended as follows

$$\phi(\mathbf{z}) \approx \frac{1}{n\lambda}\sum_{i=1}^{n} K(\mathbf{x}_i, \mathbf{z})\mathbf{v}(i). \tag{7}$$

This is known as the Nyström extension [10], which means that the eigenvectors of the empirical kernel matrix evaluated on a finite sample set can be used as approximators to the eigenfunction of the linear operator. Interestingly, (7) is proportional to the projection of a test point computed in kernel PCA [9]. The approximation can be justified by examining the convergence of eigenvalues and eigenvectors as the number of examples increases [3, 9].

Equation (7) allows us to extrapolate the ideal kernel eigenfunction from the set of training examples to the whole data set. Suppose we are given the labeled set $X_l = \{\mathbf{x}_i\}_{i=1}^m$ and the unlabeled data $X_u = \{\mathbf{x}_i\}_{i=m+1}^n$. Then, in order to expand the ideal kernel eigenfunction to the whole data set, we can simply use the $n \times m$ extrapolation matrix $K_{nm} = [K_{ll}\ K_{lu}]^\top$, where $K_{ll} \in \mathbb{R}^{m\times m}$ is the kernel evaluated on the labeled set, and $K_{lu} \in \mathbb{R}^{m\times(n-m)}$ is that evaluated between the labeled and unlabeled samples, and multiply it with the ideal kernel eigenvectors (the re-scaling of the eigenvalues can be ignored). Note that ideally, the extrapolation is best achieved by extracting the sub-block K_{nm} from ideal kernel matrix. However, since such knowledge is not available for training, we can only turn to the empirical kernel matrix for the extrapolating purpose.

3 Propagating Ideal Kernel Eigenfunctions

As has been discussed, eigenvectors of the kernel matrix play a major role in shaping the kernel matrix. Therefore, "better" eigenvectors are expected to span a kernel matrix that gives rise to better performance. However, current approaches only rectify the spectrum of the kernel matrix while keeping the eigenvectors intact. Due to various practical factors, such as noise, choice of kernel types or parameters, and the class separability, eigenvectors of the empirical kernel matrices are almost always "contaminated" and will deviate from the ideal ones. As a result, the kernel built

upon such eigenvectors may only provide restricted discrimination. Of course the spectral transform can "correct" the imperfect kernel matrix to make it better aligned to the target. However the small degrees of freedom (compared with that of eigenvectors) make such correction limited.

Motivated from these concerns, we consider the scenario where the desired set of eigenvectors is newly computed in building the semi-supervised kernel. One possibility would be learning a parametric (such as linear) transform on the eigenvectors of the graph Lapacian for better alignment. However, eigenvectors computed from the empirical kernel matrix can be inaccurate, and some intrinsical flaws may likely survive a parametric transform to affect the alignment; on the other hand, the eigenvalue decomposition of an $n \times n$ matrix is too expensive for large problems. Therefore, we follow a more direct and efficient way that obtains the desired eigenvectors by "propagating" the eigenfunctions of the ideal kernel to the whole data set. More specifically, let $\Phi^* \in \mathbb{R}^{m\times C}$ be the eigenvectors of the ideal kernel matrix K^* (each column is one eigenvector), and $\Phi \in \mathbb{R}^{n\times C}$ be the desired eigenvectors that will be used to span the ultimate kernel. Then we want to seek an extrapolation matrix $\mathbf{T} \in \mathbb{R}^{n\times m}$ such that the desired eigenvectors can be obtained via $\Phi = \mathbf{T} \cdot \Phi^*$. Here, a simple choice of the extrapolation matrix $\mathbf{T}$, as has been discussed in Section 2, is the sub-block $K_{nm} = [K_{ll}\ K_{lu}]^\top$. We have the following proposition on the property of such extrapolation.

Proposition 2. *Let the extrapolation matrix be $\boldsymbol{T} = K_{nm}$. Then the resultant eigenvectors will be smooth with regard to the graph structures. More specifically,*

$$\|\Phi(i,:) - \Phi(j,:)\|_F^2 \quad \leq \quad \rho \cdot \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2,$$

where $\rho = l\gamma\|\Phi\|_F^2$ is a constant.

Proof. Here we used the property $(K(\mathbf{x}_1,\mathbf{z}) - K(\mathbf{x}_2,\mathbf{z}))^2 \leq \gamma_K\|\mathbf{x}_1 - \mathbf{x}_2\|^2$. We sketch the proof for some popular kernels. For RBF kernel $K(\mathbf{x},\mathbf{z}) = \kappa(\|\mathbf{x}-\mathbf{z}\|)$, $|K(\mathbf{x}_1,\mathbf{z}) - K(\mathbf{x}_2,\mathbf{z})| \leq \kappa'(\xi)|\|\mathbf{x}_1-\mathbf{z}\| - \|\mathbf{x}_2-\mathbf{z}\|| \leq \kappa'(\xi)\|\mathbf{x}_1-\mathbf{x}_2\|$, where we used the middle value theorem in calculus and the triangular inequality, and γ can be chosen as $\max_\xi |\kappa'(\xi)|$; for linear or polynomial kernel $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^\top\mathbf{z}+\epsilon)^d$, $|K(\mathbf{x}_1,\mathbf{z}) - K(\mathbf{x}_2,\mathbf{z})| \leq \kappa'(\xi)|(\mathbf{x}_1-\mathbf{x}_2)^\top\mathbf{z}| \leq \kappa'(\xi)\|\mathbf{x}_1-\mathbf{x}_2\|\cdot\|\mathbf{z}\|$, and γ can be chosen as dR^d, with R the maximum of the two: pairwise sample distances, and sample (vector) norms. With this inequality, we have $\|\Phi(i,:) - \Phi(j,:)\|_F^2 = \|(K(i,:) - K(j,:))\cdot\Phi^*\|_F^2 \leq \sum_{k=1}^l (K(\mathbf{x}_i,\mathbf{x}_k) - K(\mathbf{x}_j,\mathbf{x}_k))^2 \cdot \|\Phi^*\|_F^2 \leq \rho\cdot\|\mathbf{x}_i-\mathbf{x}_j\|^2$. This completes the proof. □

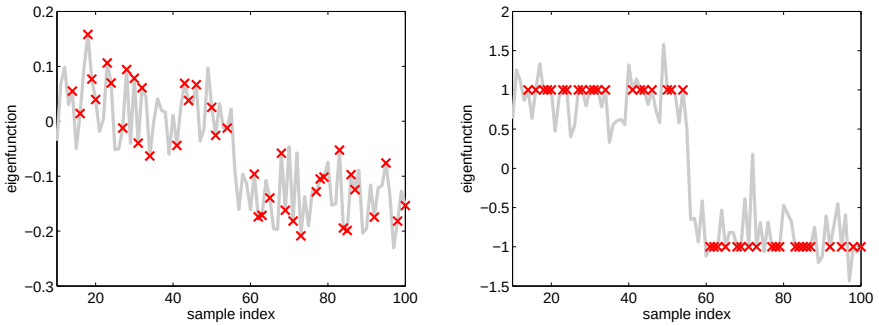
Proposition 2 indicates that, entries of the extrapolated eigenvector on two samples, $\mathbf{x}_i$ and $\mathbf{x}_j$, will have a difference that is bounded by the input space distance $\|\mathbf{x}_i - \mathbf{x}_j\|$. Therefore the eigenvector is expected to be smoothly varying with regard to the graph structures, which is a desirable property for predicting class labels. We can also divide each row of K_{nm} by the sum of the entries in that row to enforce an extra level of smoothness, which prevents the numerical fluctuations of the eigenvector.

Note that the extrapolation not only extends the eigenfunction to the unlabeled samples, but also simultaneously "re-estimates" the eigenfunction on the training samples. Therefore, a natural requirement for the extrapolation is that the reconstructed eigenfunction should be consistent with the ideal kernel eigenfunction (on the training samples), which is similar in spirit to kernel target alignment. To achieve this, we introduce an extra linear transform $P \in \mathbb{R}^{l\times l}$ on the extrapolation matrix K_{nm}, i.e., $\mathbf{T} = K_{nm}P$. This linear transform imposes a global, consistent distortion on K_{nm} for both the labeled and unlabeled set of samples. Based on the transform $\mathbf{T}$, the extrapolated eigenvectors on the labeled set can be written as $\Phi_{tr} = K_{ll}P\Phi^*$. In order to make Φ_{tr} and Φ^* close, obviously, we should choose $P = K_{ll}^{-1}$. In practice, to avoid numerical instabilities, we choose $P = (K_{ll}+\sigma\mathbf{I})^{-1}$, where $\mathbf{I}$ is the identity matrix and σ is a small positive number. Then, the desired eigenvector (on all the data points) can be obtained as

$$\Phi = \begin{bmatrix} K_{ll} \\ K_{ul} \end{bmatrix} (K_{ll} + \sigma\mathbf{I})^{-1} \cdot \Phi^*, \tag{8}$$

and the corresponding kernel matrix is $\Phi\Phi^\top$. Here, for simplicity we set all the C eigenvalues to be unity, since as is shown in Section 2, the most important discriminative information has already been encoded in the eigenvectors. It's interesting to note that equation (8) is in the form of kernel ridge regression, i.e., the labeled set $\mathcal{X}_l$ is used as the training samples to predict the unlabeled points.

We illustrate the effect of the normalization $(K_{ll} + \sigma\mathbf{I})^{-1}$ on the eigenfunction extrapolation. We choose 100 USPS digits (the first 50 are "4" and next 50 are "9") and randomly choose 30 for



(a) without normalization (b) with normalization.

Figure 1: Effect of the normalization term on the extrapolation of eigenfunctions. Gray curve are extrapolated eigenvector and red crosses denote labeled samples. By normalization, entries of the eigenfunction belongs to the same classes are "squeezed" to the same value; while those for different classes are pushed apart.

labeling. Then we extend the ideal kernel eigenvector $\Phi^* \in \mathbb{R}^{30\times 1}$ to $\Phi \in \mathbb{R}^{100\times 1}$. Figure 1 shows the result with and without the normalization. As can be seen, direct extrapolation using $\mathbf{T} = K_{nm}$ may not guarantee that the extrapolated entries have similar values for samples in the same class; using the extrapolation $\mathbf{T} = K_{nm}(K_{ll} + \sigma\mathbf{I})^{-1}$ can guarantee that the reconstructed eigenfunction has almost the same value as the class labels on the training set, and the discrimination between classes is also improved.

With the introduction of the normalization term, the smoothness property in Proposition 2 still holds, but the constant ρ will be re-scaled by the Frobenius norm $\|P\|_F = \|(K_{ll} + \sigma\mathbf{I})^{-1}\|_F$. As can be seen, the regularization imposed by $\sigma\mathbf{I}$ can bound the magnitudes of the smoothness factor, or else $\|K_{ll}^{-1}\|_F$ will approach infinity if K_{ll} is singular. Actually, this also helps "strengthen" the connection between the eigenfunction on the labeled and unlabeled samples, which guarantees that a good alignment on the training set can transfer to that of the testing samples. In the extreme case $\sigma \to +\infty$, $\mathbf{T}$ will be the same as K_{nm} (up to a scaling difference). The specific level of "concentration" of alignment will depend on the extrapolation matrix $\mathbf{T}$ applied, and a rigorous theoretical analysis will be provided in the future to shed light on such relationship. In practice, we simply fix $\sigma = 10^{-3}$ in all our experiments, which gives consistently good performance.

At last we want to emphasize the computational efficiency of our method. Note that the number of eigenvectors needed in our algorithm equals the number of classes. In comparison, the number of eigenvectors used in spectral transformations is usually chosen manually and much larger. On the other hand, our algorithm needs not perform the eigenvalue decomposition of the $n \times n$ kernel matrix, thereby avoiding the computational bottleneck for the whole family of algorithms based on spectral transforms. The time complexity of our method is $O(nmC + m^3)$, where n is the sample size, m the training set size, and C the number of classes (the expansion of the ultimate kernel matrix that takes $O(n^2)$ time is common for all semi-supervised kernel design schemes if classification is performed). Empirically, our algorithm is tens to hundreds of times faster than the counterparts.

4 Experiments

In this section, we compare the following semi-supervised kernels: (1) cluster kernel [1], where $r(\cdot)$ is chosen as linear function $r(\lambda) = \lambda$; (2) diffusion kernel $r(\lambda) = \exp(-\lambda/c)$ [4]; (3) gaussian field kernel $r(\lambda) = \frac{1}{\lambda+\epsilon}$ [8]; (4) our approach; (5) non-parametric graph kernel [5] using the first $p = 0.1n$ eigenvectors from the Laplacian $\tilde{\mathcal{L}}$. Evaluation is based on the alignment on the test set and the classification performance of standard SVM using the learned kernel. We tested 21 data sets from some machine learning benchmark data sets[1]. We used the Gaussian kernel $G(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 \cdot b)$ in all our experiments. The parameters are chosen as follows. For the kernel width, we first compute b_0 as the inverse of the average squared pairwise distances, and then choose

[1] semi-supervised learning www.kyb.tuebingen.mpg.de/ssl-book/; libsvm data www.cse.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

Table 1: Classification Performance semi-supervised kernel design schemes. For each cell, the top row is the mean/std of the kernel alignment score (in $[0, 1]$) on the test set, and in bracket is the averaged time consumption (in seconds); the bottom row is the mean/std of classification error (%).

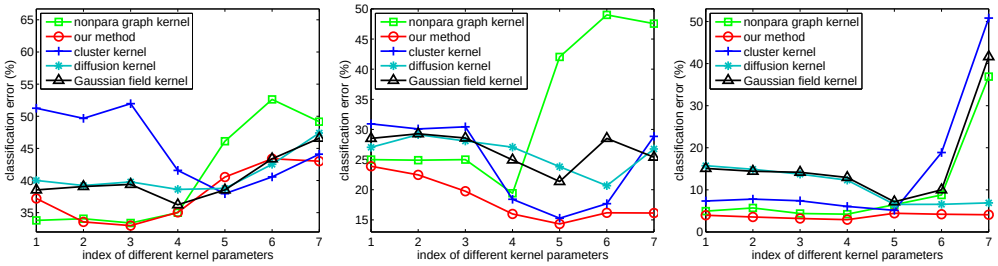
Data size/dim	Non-paramatric Graph Kernel	Ours	Cluster Kernel linear	Diffusion Kernel	Gaussian-field Kernel
Digit1	0.29±0.07 (84.9)	0.82±0.02 (0.3)	0.13±0.005 (2.4)	0.10±0.001 (13.0)	0.14±0.001 (12.6)
1500×241	4.31±1.93	4.89± 0.85	5.37±1.23	6.13±1.63	**3.82±1.23**
USPS	0.23±0.08 (74.9)	0.66±0.04 (0.3)	0.43±0.001 (2.5)	0.06±0.001 (16.0)	0.06±0.01 (12.7)
1500×241	7.47± 4.41	**6.64±1.27**	**6.56±1.02**	7.27±0.59	9.81±0.49
COIL2	0.11±0.005 (73.4)	0.55±0.07 (0.3)	0.10±0.001 (2.4)	0.05±0.003 (8.4)	0.07±0.00 (5.3)
1500×241	18.49±2.47	**13.44±2.41**	18.51±4.66	19.08±2.05	19.32±1.89
BSI	0.07±0.003 (9.9)	0.14±0.04 (0.1)	0.04±0.001 (0.2)	0.07±0.003 (0.4)	0.07±0.002 (0.5)
1500× 241	**32.95±3.38**	**32.99±3.10**	42.02±2.89	33.58±2.83	34.85±2.75
COIL	0.01±0.001 (199.5)	0.11±0.005 (x)	0.08±)0.002 (2.58)	0.06±0.001 (8.3)	0.07±0.001 (5.5)
1500×241	21.90±3.24	**9.14±0.96**	10.89±1.12	11.67±1.43	11.75±1.49
g241n	0.40±0.003 (108.2)	0.33±0.03 (0.3)	0.03±0.007 (2.5)	0.04±0.00 (20.3)	0.04±0.00 (6.7)
1500×241	**13.64±1.28**	24.11±1.73	26.59±3.96	19.68±1.52	18.61±1.75
Text	0.13±0.01 (181.0)	0.30±0.02 (4.71)	0.03±0.001 (68.1)	0.03±0.00 (208.0)	0.03±0.004 (130.7)
1500×11960	25.55±1.65	**23.42±1.46**	32.90±6.64	24.89±1.81	26.78±4.88
usps38	0.48±0.004 (77.3)	0.84±0.02 (0.3)	0.12±0.001 (1.6)	0.11±0.001 (6.8)	0.11±0.001 (4.5)
1200×256	4.82±1.33	**2.82±0.83**	5.10±0.89	6.06±1.01	6.06±0.85
usps49	0.40±0.13 (82.1)	0.86±0.01 (0.3)	0.09±0.001 (1.9)	0.08±0.001 (9.3)	0.07±0.001 (8.9)
1296×256	2.83±0.92	**1.98±0.52**	6.29±2.11	8.26±0.83	10.67±1.24
usps56	0.48±0.06 (80.0)	0.86±0.01 (0.3)	0.12±0.001 (1.7)	0.09±0.003 (18.2)	0.11±0.001 (5.0)
1220×256	2.87±0.92	**2.44±0.59**	3.89±1.57	3.85±0.97	5.79±1.06
usps27	0.58±0.004 (101.8)	0.91±0.006 (0.3)	0.37±0.001 (2.3)	0.10±0.001 (11.8)	0.13±0.001 (6.9)
1376×256	1.79±0.42	**1.21±0.25**	1.80±0.25	2.28±0.56	4.80±1.29
odd/even	0.21±0.008 (419.0)	0.65±0.03 (0.4)	0.12±0.001 (8.8)	0.03±0.004 (38.5)	0.08±0.00 (22.3)
2007×256	10.14±2.11	**9.58±1.56**	14.59±1.49	14.08±2.04	15.64±2.91
adult1a	0.28±0.02 (116.7)	0.36±0.04 (0.3)	0.29±0.001 (2.0)	0.25±0.002 (13.9)	0.04±0.00 (18.9)
1605×123	26.90±1.75	**23.84±2.13**	24.35±1.98	31.39±2.56	31.89±3.00
w1a	0.22±0.11 (155.4)	0.34±0.08 (0.3)	0.07±0.00 (3.8)	0.12±0.00(31.2)	0.17±0.00 (22.6)
2477×300	24.78±8.97	20.45±6.17	46.33±28.06	16.98±9.32	**8.07±2.49**
sonar	0.17±0.03 (3.7)	0.57±0.07 (0.1)	0.10±0.01(0.1)	0.10±0.004 (0.2)	0.10±0.004 (0.4)
208×60	21.11±3.26	14.57±3.24	15.88±3.74	**14.16±3.06**	22.70±3.62
australian	0.39±0.07 (7.53)	0.55±0.02 (0.2)	0.02±0.00 ()	0.16±0.01 (1.5)	0.16±)0.004 (1.4)
690×14	14.56±0.67	**13.19±0.64**	14.45±1.21	18.48±2.92	14.93±1.31
diabetes	0.16±0.06 (34.75)	0.28±0.03 (0.1)	0.17±0.003 (0.3)	0.05±0.001 (4.8)	0.06±0.001 (1.5)
768×8	26.14±1.82	**25.67±1.87**	27.20±1.64	28.84±1.74	29.95±2.02
splice	0.13±0.004 (53.6)	0.32±0.03 (0.1)	0.02±0.003 (0.3)	0.002±0.00 (4.2)	0.05±0.00 (4.1)
1000×60	**23.75±2.55**	24.49±1.85	26.53±2.58	24.48±1.48	24.17±1.46
german	0.03±0.01 (27.5)	0.19±0.04 (0.1)	0.20±0.00 (0.3)	0.19±0.003 (0.6)	0.04±).01 (2.5)
1000×24	32.54±3.92	**30.94 ±2.48**	32.24±3.31	36.99±3.47	36.86±3.21
svmguide1a	0.28±0.01 (436.3)	0.68±0.14 (0.2)	0.23±0.03 (4.6)	0.07±0.00 (86.7)	0.10±0.001 (38.9)
3089×4	5.82±1.80	**5.10±0.97**	5.73±0.51	6.24±1.66	7.13±1.75
liver	0.07±0.007 (11.0)	0.12±0.03 (0.1)	0.03±0.01 (0.1)	0.04±0.00 (0.6)	0.06±0.002 (0.5)
345×6	33.67±3.85	**32.23±3.69**	38.14±3.95	38.46±3.16	37.69±3.64

b among $b_0 \cdot \{\frac{1}{25}, \frac{1}{10}, \frac{1}{5}, 1, 5, 10, 25\}$ that gives the best performance. The parameter c and ϵ are chosen from $\{10^{-5}, 10^{-3}, 10^{-1}, 1\}$ and $\{10^{-2}, 10^{-1}, 1, 10, 10^2\}$, respectively. For each data, the algorithms are repeated 30 times with 50 labeled samples randomly chosen for each class. Results are reported in Table 1, including time consumption, alignment score, and classification error.

As can be seen from Table 1, our algorithm gives competitive performance on most data sets. It is also the most efficient and can be hundreds of time faster those algorithms based on parametric or nonparametric transforms. We note that the cluster kernel using linear spectral transform is also quite efficient, because it only needs to compute the normalized kernel matrix but not the eigenvalue decomposition. However the performance is inferior. Our algorithm gives high alignment score for most of the data sets. On some data sets (e.g. splice), our algorithm has a lower prediction accuracy but higher alignment. This is because the classifier used (SVM) involves complex optimization while the connection between kernel alignment and the generalization performance is currently only shown for simple classifiers like the Parzen window [2].

We observe that algorithms using spectral transforms are relatively sensitive to the kernel parameter. Although the kernel target alignment can somewhat "correct" the imperfect kernel matrix, the small degrees of freedom of the eigenvalues (compared with eigenvectors) may limit such corrections. In comparison, our approach compute the desired eigenvectors via propagating the ideal kernel eigenfunction and is therefore more robust. Figure 2 plots the classification perfor-

mance of different semi-supervised kernels w.r.t. the RBF kernel parameter b (from 7 grid values $b_0 \cdot \{\frac{1}{25}, \frac{1}{10}, \frac{1}{5}, 1, 5, 10, 25\}$). As can be seen, the performance of our method is relatively stable. This greatly eases the difficulty of parameter tuning: in practice our algorithm gives reasonably good performance by simply choosing the RBF kernel width as the averaged square pairwise sample distances (i.e., $b = b_0$), which can be observed from Figure 2 as well.



(a) liver disorder (b) sonar (c) usps3/8

Figure 2: Performance of different semi-supervised kernels w.r.t. the kernel parameter.

5 Conclusion

This paper proposed a new algorithm for semi-supervised kernel design. Unlike traditional methods that directly use the eigenvectors of the graph Lapacian with a rectified spectrum, we propose to compute the desired eigenvectors via propagating the ideal kernel eigenfunction. The key observation is that, while eigenvectors from the empirical kernel matrix alone may contain inevitable flaw due to various practical factors, computing them based on extrapolating the ideal kernel eigenfunction will be more reliable and less dependent on the choice of empirical kernels. The extrapolating process reflects important similarity relation encoded in the input data, and also takes into account the alignment with the target concept. Our algorithm gives encouraging performance and the time complexity is only linear in the same size. In the future we shall explore different ways for propagating the ideal kernel information and combination of multiple kernels from different resources.

References

[1] O. Chapelle, J. Weston and B. Schölkopf: Cluster Kernels for Semi-Supervised Learning. Advances in Neural Information Processing Systems 15, 2003, 585-592..

[2] N. Cristianini, J. Shawe-taylor, A. Elissee, J. Kandola. On kernel-target alignment. Advances in Neural Information Processing Systems 14, 2002, 367–373.

[3] J. Shawe-Taylor, C. Williams. The stalibity of kernel pricipal component analysis and its relation to the process eigenspectrum. In Advances in Neural Information Processing Systems 15.

[4] R.I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In Proc. 19th International Conf. on Machine Learning, 2002.

[5] X. Zhu, J. Kandola, Z. Ghahbramani and J. Lafferty. Nonparametric Transforms of Graph Kernels for Semi-Supervised Learning. In Advances in Neural Information Processing Systems 17, 2004.

[6] G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. Jordan. Learning the kernel matrix with semidefinite programming. Journal of Machine Learning Research, 5:27–72, 2004.

[7] A. Smola and R. Kondor. Kernels and regularization on graphs. In Conference on Learning Theory, COLT/KW, 2003.

[8] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In 20th International Conference on Machine Learning, 2003.

[9] Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-Franois Paiement, Pascal Vincent, Marie Ouimet. Learning Eigenfunctions Links Spectral Embedding And Kernel Pca (2004), Neural Computation Volume 16(10):2197 - 2219.

[10] C. Williams, M. Seeger. Using the Nyström Method to Speed Up Kernel Machines. Advances in Neural Information Processing Systems 13, 2001.